What, if anything, can be done in linear time?

Yuri Gurevich

Computer Science & Engineering
University of Michigan

June 30, 2020, St. Petersburg via Zoom

Agenda

1. Linear time computation complexity
2. Basic primal logic
3. Extensions

1. Linear time computation complexity

1.1. Computation model

Why not polynomial time?

Practicality.

Even linear time may be excessive:

1. The debate with Steve Cook.
2. Counting unique queries.

What linear time?

The linear time of the standard computation model of the analysis of algorithms, the RAM. More details below.

Term "random access" is misleading.
The important part is this:
constant-time memory access.

Sorting

Sorting n items requires $\Omega(n \log n)$ comparisons and thus $\Omega(n \log n)$ time.

There is no way around that lower bound.
Or maybe there is?

Recall arrays

An array A:

- Indices: $0, 1, \ldots, n-1$
- Values: $A[0], A[1], \ldots, A[n-1]$

"Data structure" is another misleading term.
An array A is also an algorithm which, given an index i, produces $A[i]$ *in constant time*.

Theorem: Arrays of natural numbers $< n$ can be sorted in time $O(n)$.

`Illustration.` Given $A = \langle 3, 6, 0 \rangle$,

1. Create (i.e. allocate space for) an auxiliary array B of length 7, and then zero it: $B = \langle 0, 0, 0, 0, 0, 0, 0 \rangle$.
2. Traverse A setting every $B[A[k]]$ to 1. Now $B = \langle 1, 0, 0, 1, 0, 0, 1 \rangle$.
3. Traverse B and output the indices of value 1 into a new array: $\langle 0, 3, 6 \rangle$.

The random access machine

Let n be length of the input.

- Only polynomially many initial registers are used, with addresses and registers of length $O(\log n)$.
- Value(address) is read and written in constant time.
- Arithmetical relations $=, \leq, \geq$ and operations $+, -$ are constant time.

The model reflects the standard computer architecture.

It is often assumed that multiplication and division are also constant time; we will not need that.

1.2. Homonymy originals

Suffix arrays

The *suffix array* for a string $c_0, \ldots, c_{n-1}$ of characters is an array A of length n where $A[i]$ is the j such that $c_j \ldots c_{n-1}$ is the i^{th} suffix in the lexicographical order. Think of this j as the key for the suffix $c_j \ldots c_{n-1}$. In an appropriate sense, the suffix array orders the suffixes lexicographically.

There is an amazing algorithm that constructs suffix arrays in linear time.

A useful linear-time parsing algorithm

1. Use a deterministic pushdown automaton to produce the parse tree of a given formula φ.
2. Construct the suffix array for φ.
3. Traverse the suffix array and construct pointers $H(u)$ from every node u to the first node with the same subformula.

$H(u)$ is the *homonymy original* of u and the representative of the subformula of u.

2. Basic primal logic

Main reference:

Carlos Cotrini and YG, "Basic primal infon logic"
Journal of Logic and Computation 26:1 (2016)
(#215 at YG's website)

2.1. Motivation

Infon algebra

Primal (infon) logic was introduced in 2009 in the framework of policy and trust management, but here we motivate it from first principles.

Infon is an item of information, not necessarily true or false. A meaningful question is whether it is known to an agent.

$y \leq x$ means: y is at least as informative as x.

Conjunction turns the preorder into a semilattice. In addition to the binary connective $\wedge$, there are also unary connectives, one for each agent p:

$$p \text{ said } x$$

Infon algebra to infon logic

Implication $a \to b$ is a solution for $(a \wedge x \leq b \leq x)$.
(Requiring a greatest solution leads to intuitionistic implication.)

Disjunction $a \vee b$ is a solution for $(a \leq x$ and $b \leq x)$.
(Requiring a smallest solution leads to intuitionistic disjunction [Beklemishev-YG]).

The resulting basic primary logic has a remarkable combination of expressivity (e.g. of typical access-control scenarios) and feasibility (to be discussed).

2.2. Derivation

Derivation rules

For expository reasons, we restrict attention to the conjunction and implication fragment.

$$\frac{x \wedge y}{x} \qquad \frac{x \wedge y}{y} \qquad \frac{x, y}{x \wedge y}$$

$$\frac{x, x \to y}{y} \qquad \frac{y}{x \to y}$$

Subformula property

Theorem

If $\alpha_1, \ldots, \alpha_\ell$ is a shortest derivation of φ from H then every α_i is a subformula of (a formula in) $H \cup \{\varphi\}$.

Interpolation lemma

Lemma

If $H \vdash \varphi$, then there is a set I of subformulas of H that are also subformulas of φ such that

1. *all I formulas are derivable from H, and*
2. *φ is derivable from I via introduction rules only.*

We will not use the interpolation lemma but it gives a useful optimization in the case where the hypotheses change rarely.

Multiderivation problem

Definition

Given sets H of hypotheses and Q of queries, decide which of the queries follow from the hypotheses.

2.3. Decision algorithm

Theorem

The multiderivation problem for propositional infon logic is solvable in linear time.

Our approach is: Derive them all!
Compute all subformulas of $H \cup Q$ derivable from H.

Initial condition

Initially,

- all subformulas of $H \cup Q$ are *raw*,
- the hypotheses are *pending*, and
- no formulas are *processed*.

Pseudocode

Repeat until no formula is pending.

1. Pick the first pending formula α.
2. Apply all possible inference rules to α, and if a newly derived formula is raw then mark it pending.
3. Mark α itself processed.

One easy case

Apply the first $\wedge$-elimination rule $\dfrac{x \wedge y}{x}$ to $\alpha = \alpha_1 \wedge \alpha_2$ and thus derive α_1.

If α_1 is raw then mark it pending.

One harder case

Apply the $\wedge$-introduction rule $\dfrac{x, y}{x \wedge y}$ to α where α plays the role of x.

If $\alpha \wedge y$ is raw but y is pending or processed, mark $\alpha \wedge y$ pending.

But how do we find the relevant formulas y? We don't have time to walk through the raw formulas over and over again.

Local search

Traverse the parse tree for $H \cup Q$ and construct *use lists* $(\wedge, \ell)$, $(\wedge, r)$, $(\to, \ell)$, $(\to, r)$ for every homonymy original node u.

$(\wedge, \ell)$ If node $u = H(u)$ is the left child of a node w with subformula $x \wedge y$, put $H(w)$ into the use set $(\wedge, r)$ of u. (Notice that the subformula of u is x.)

Similarly for $(\wedge, r), (\to, \ell)$ and $(\to, r)$.

Back to applying $\dfrac{x, y}{x \wedge y}$ to $x = \alpha$

Recall: we are looking for formulas $\alpha \wedge y$, more exactly for the nodes representing these formulas.

Let $u = H(u)$ represent α.

Just walk through the use set $(\wedge, \ell)$ of u.

3. Extensions

3.1. Disjunctions

Original motivation

Rules of the form

$$\texttt{if } \alpha \texttt{ then ACTION}$$

are common in access control. Suppose that $\alpha = \alpha_1 \vee \alpha_2$, e.g.

$$\mathrm{passport}(\mathrm{US}) \vee \mathrm{passport}(\mathrm{UK}).$$

Such disjunctions may be eliminated but they may make the rule exponentially more succinct.

New rules

Add introduction rules

$$\frac{x}{x \vee y} \qquad \frac{y}{x \vee y}.$$

The linear-time decision algorithm generalizes in a rather obvious way.

3.2. Transitive primal logic

Reference:

Carlos Cotrini and YG
"Transitive primal infon logic: the propositional case"
Review of Symbolic Logic 6:2 (2013)
(#211 at YG's website)

3.3. Conjunctions as sets

Reference:

Carlos Cotrini, YG, Ori Lahav, and Artem Melentyev
"Primal infon logic with conjunctions as sets"
Springer LNCS 8705 (2014)
(#221 at the YG website)

Motivation

While $x \wedge y$ entails $y \wedge x$ in basic primal logic,

- $(x \wedge y) \to z$ does not entail $(y \wedge x) \to z$,
- $z \to (x \wedge y)$ does not entail $z \to (y \wedge x)$,
- etc.

Idea, a problem and proposed solution

Idea: View conjuncts as sets.

Problem: Sets are not constructive objects.

Proposed solution: Represent sets as sequences by ordering conjuncts lexicographically.

Decision algorithm

The resulting multiderivation problem is solvable in expected linear time.

It is the algorithm that introduces randomness.
No probability distribution on inputs is assumed.